Blue Gene/L: Applications and Tracing

Maciej Brodowicz

Sharon Brunett

Caltech

Magnetic Hydro Dynamic (MHD) Code Overview

- Parallel MHD fluid code solves equations of hydrodynamics and resistive Maxwell's equations
  - Part of larger application which computes dynamic responses to strong shock waves impinging on target materials
  - Fortran 90 + MPI
  - MPI Cartesian communicators
  - Nearest neighbor comms use non blocking send/recv
  - MPI Allreduce for calculating stable time steps

MHD Parallel Implementation

- Physical domain decomposed into regular domains assigned to processors
  - Extra cells surround domain, yielding overlap region between neighboring subdomains
  - Nearest neighbor comms exchange data in overlapping regions
  - Global reduction of minimum timestep at the start of each timestep

MHD Trace Summary

- Blute[convert,merge] used to post process input to BG/L network simulator
- 100 iteration runs, 16 - 512 CPUs using blsim2.0
  - 161 MB merged trace.ute file from 256 CPU
  - Blfakemain statistics on 256 way blute input file
    - 768,000 simulated MPI Isend/Irecv events
    - Min msg size = 320 bytes, max = 25600 bytes, avg = 1720 bytes
    - 8x8x4 torus => 2.83 hops / pnt. 2 pnt. message
    - 4x8x8 torus => 1.63 hops/ pnt. 2 pnt message

MHD Summary

- Traces through network simulator show code is CPU bound
  - Increasing problem size doesn’t increase comms to computation ratio
    - Good for production runs, bad for network stress testing!
- MHD’s good scaling, simple MPI calls and CPU bound characteristics make it a good candidate for node simulator tracing
  - Anticipating progressively useful node simulator binaries from IBM in Sept.
    - Functionality verification, review assembly output, cache level statistics for main MHD loops

Quantum Monte Carlo (QMC) Application

- Computational method for potentially allowing material properties to be calculated to within chemical accuracy
  - Code base from ASCI/ASAP Material Properties group at Caltech, led by Bill Goddard
  - Manager/worker paradigm
    - Two computationally intensive phases with statistics gathering MPI_Reduce done on manager node
  - Efficient parallel algorithm being developed to efficiently divide calculation among available processors and minimize global communication

QMC Tracing Status

- Ported to SP2 (blue pacific) and SP3 (frost) at LLNL
  - C++ code, works with native and GNU compilers
- Trace files (raw and ute) completed for 128 way runs
  - Blfakemain post processing of merged traces causes core dumps

QMC Summary

- Network traces for QMC use M. Brodowicz’s trace analysis scripts
  - MPI_Reduce and MPI_Iprobe counts were trivial, compared to computational demands
- QMC test kernel for distributing workload to many workers with smaller memory working on SP2
  - Ready to test on IBM’s node simulator

Gyrokinetic Toroidal Code (GTC) Overview

- GTC calculates micro-turbulence in a tokamak using kinetic equations in a collisionless regime
  - Developed at Princeton Plasma Physics Lab. by Stephane Ethier
  - Sources are Fortran 90 + MPI
- Torus geometry surrounded by twisting magnetic field lines
  - Grid follows magnetic field lines in real space
  - Particles move around the torus, along magnetic field lines, at very high velocities yet have a much slower motion in the perpendicular direction
  - One dimensional domain decomposition in toroidal direction

GTC Implementation and Tracing

- Particle in Cell (PIC) approach
- Particles cross boundaries of two domains via MPI_sendrecv nearest neighbor calls
- Highly compute bound except for synchronizing MPI_Allreduce calls
  - 95% compute bound according to BLUTE2.1 traces

GTC Tracing Summary

- Production code not a good network stress test
  - 64 way GTC run yields 832,128 MPI_Gather events and 280 Gcycles burst per node
  - 96 way GTC run for fixed problem size becomes less interesting
    - 94,080 MPI_Gather events and 930 Gcycles burst per node
- Eliminating turbulence calculations and electric field fluctuations yields better network stress test
  - 15% comms overhead on 96 way run

GTC Event Count and Message Sizes
64 CPUs 200 Iterations

	MPI_Allreduce	MPI_Sendrecv	MPI_Bcast
Count	57,753	359,074	3,265
Min [bytes]	25,779,005	25,729,324	22,758
Max [bytes]	357,552,832	357,849,187	30,849,074
Avg. [bytes]	192,993,517	193,211,951	22,146,817

3-D Adaptive Mesh Refinement (AMR3D)

Fluid dynamics code based on two major software components:

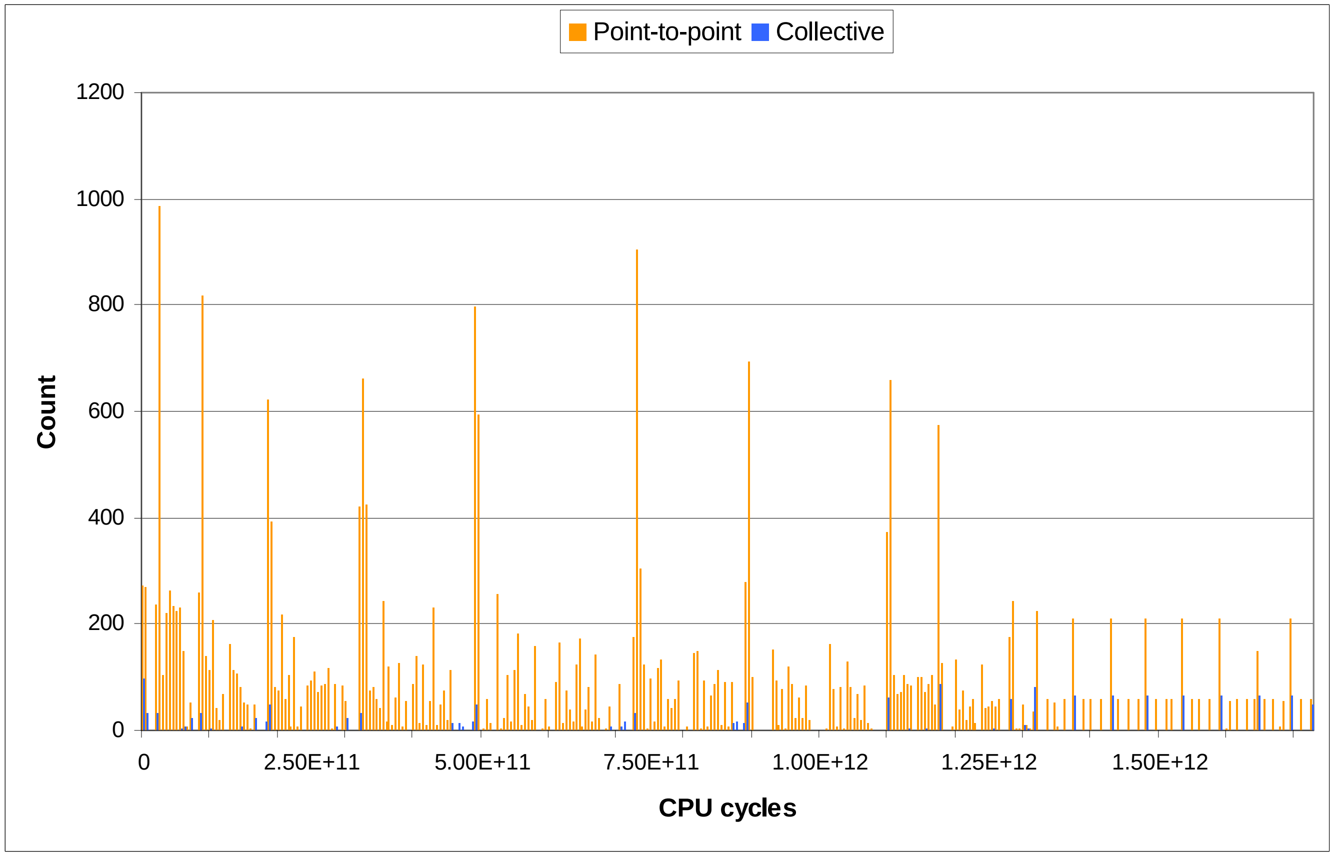
- Richtmyer-Meshkov shock simulation with Cartesian meshes written by Ravi Samtaney (Caltech, Princeton) – Fortran
- GrACE data management library designed by Manish Parashar (Rutgers University) – C++. Grace supports mesh generation and removal, automatic load balancing and runtime statistics gathering.

AMR3D’s behavior is highly dynamic with respect to CPU utilization and memory usage. Communication patterns include:

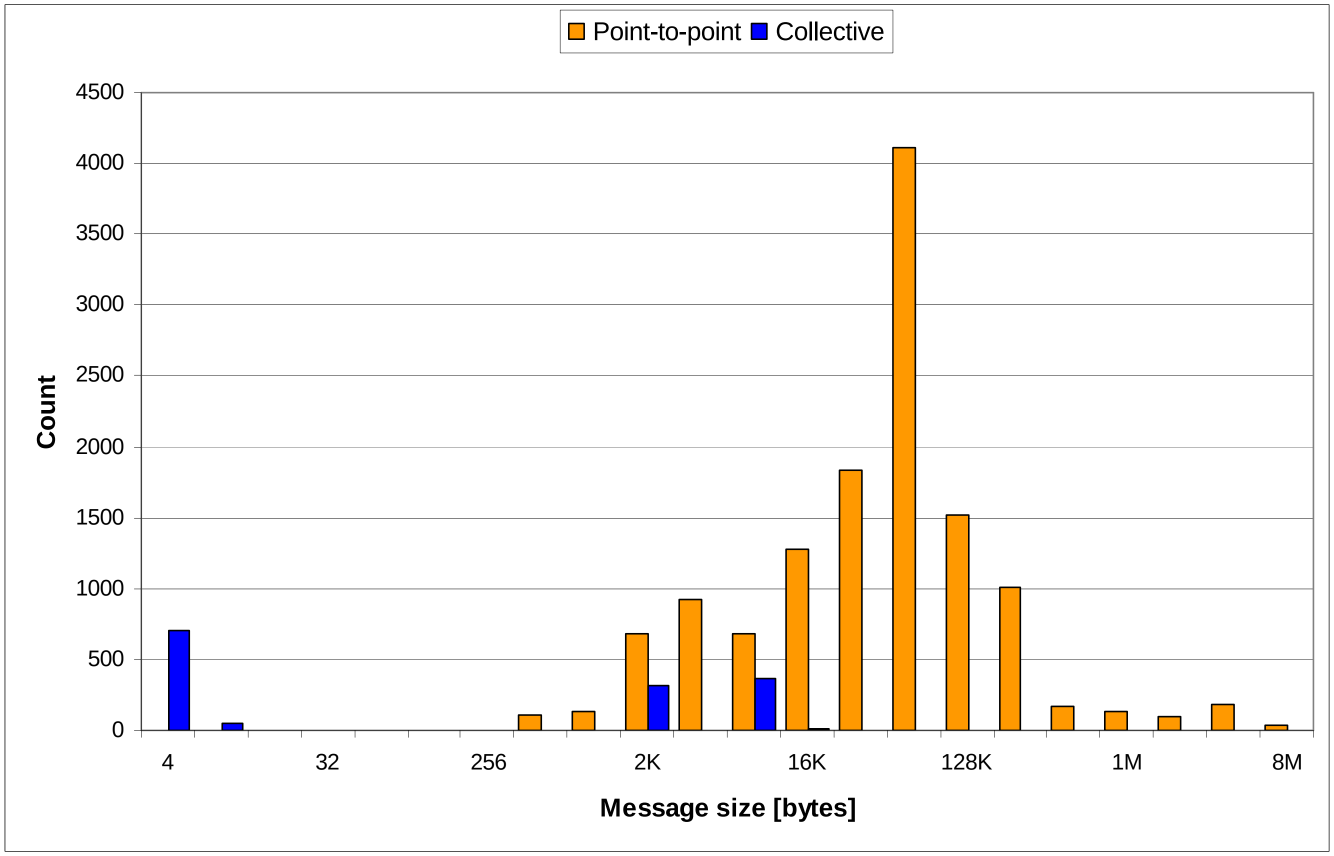
- Point-to-point nearest-neighbor updates of boundary regions
- Global reductions to determine the timestep value to be used throughout the next iteration
- Collective and point-to-point message bursts generated by grid recomposition events

AMR3D: Communication Profile

50 time steps on 16 processors, initial grid size 2048x32x32



AMR3D: Message Size Distribution



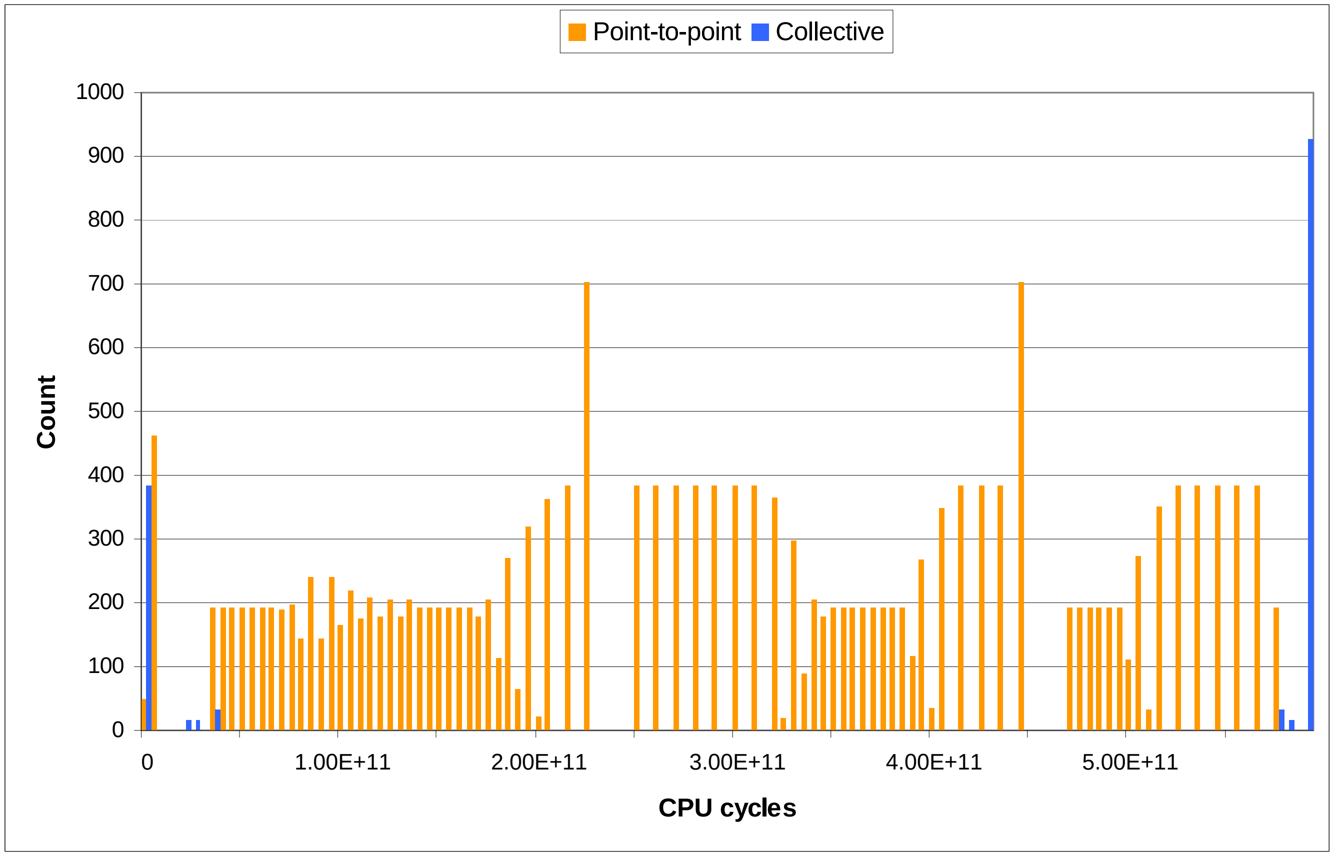
Lennard-Jones, Spatial Decomposition (LJS)

Short-range molecular dynamics code written in Fortran by Steve Plimpton (SNL):

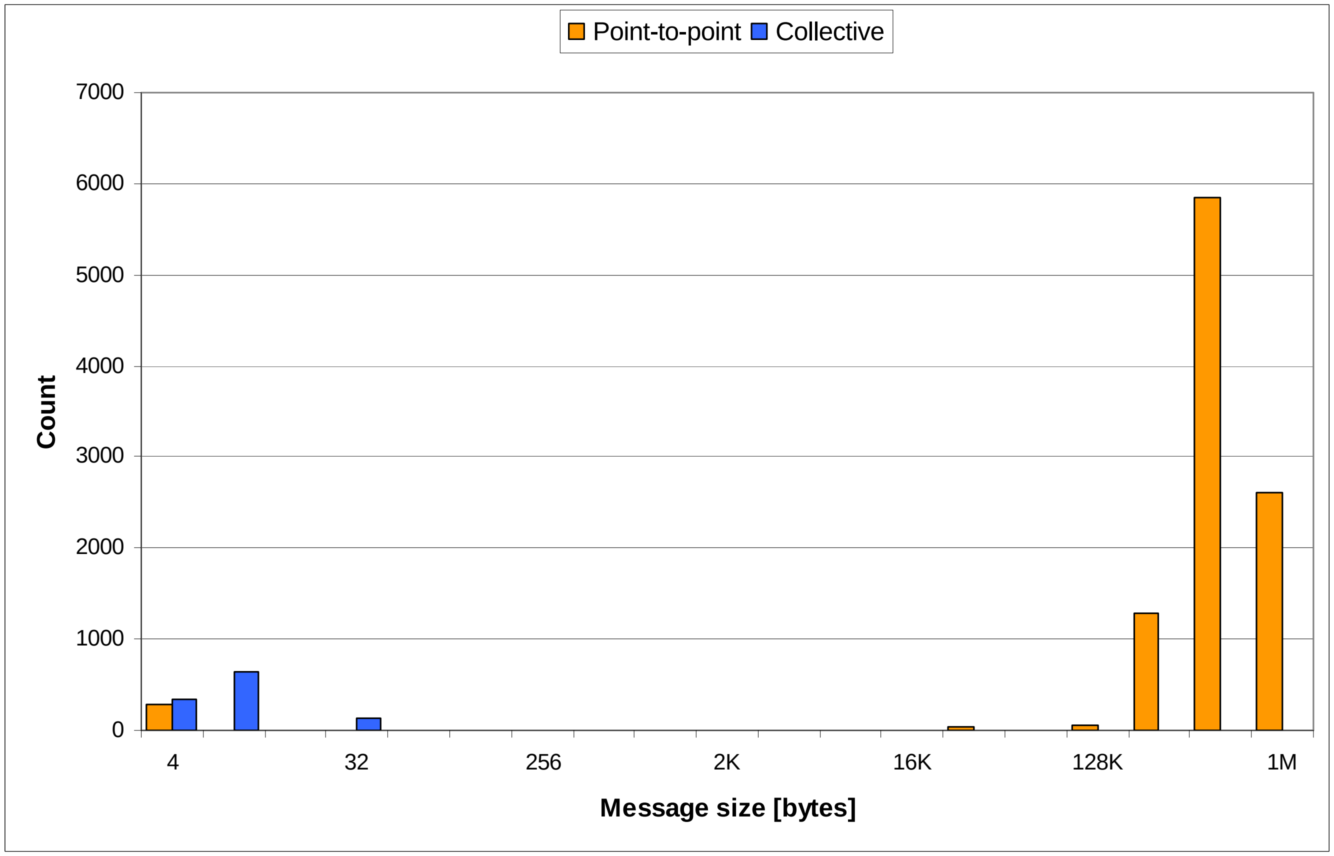
- Simulates Newtonian interactions in large groups of atoms
- Each processor keeps track of the positions and movements of atoms in its 3-D “box”
- Point-to-point messages are exchanged:
  - At the end of each timestep to acquire atom positions from nearby “boxes” (for computation of forces)
  - Every few timesteps to reassign positions of atoms due to movement (binning)
- Collective calls used exclusively during setup phase and computation of the final statistical information

LJS: Communication Profile

50 time steps on 16 processors, problem size 160^3 (16 mil. atoms)



LJS: Message Size Distribution



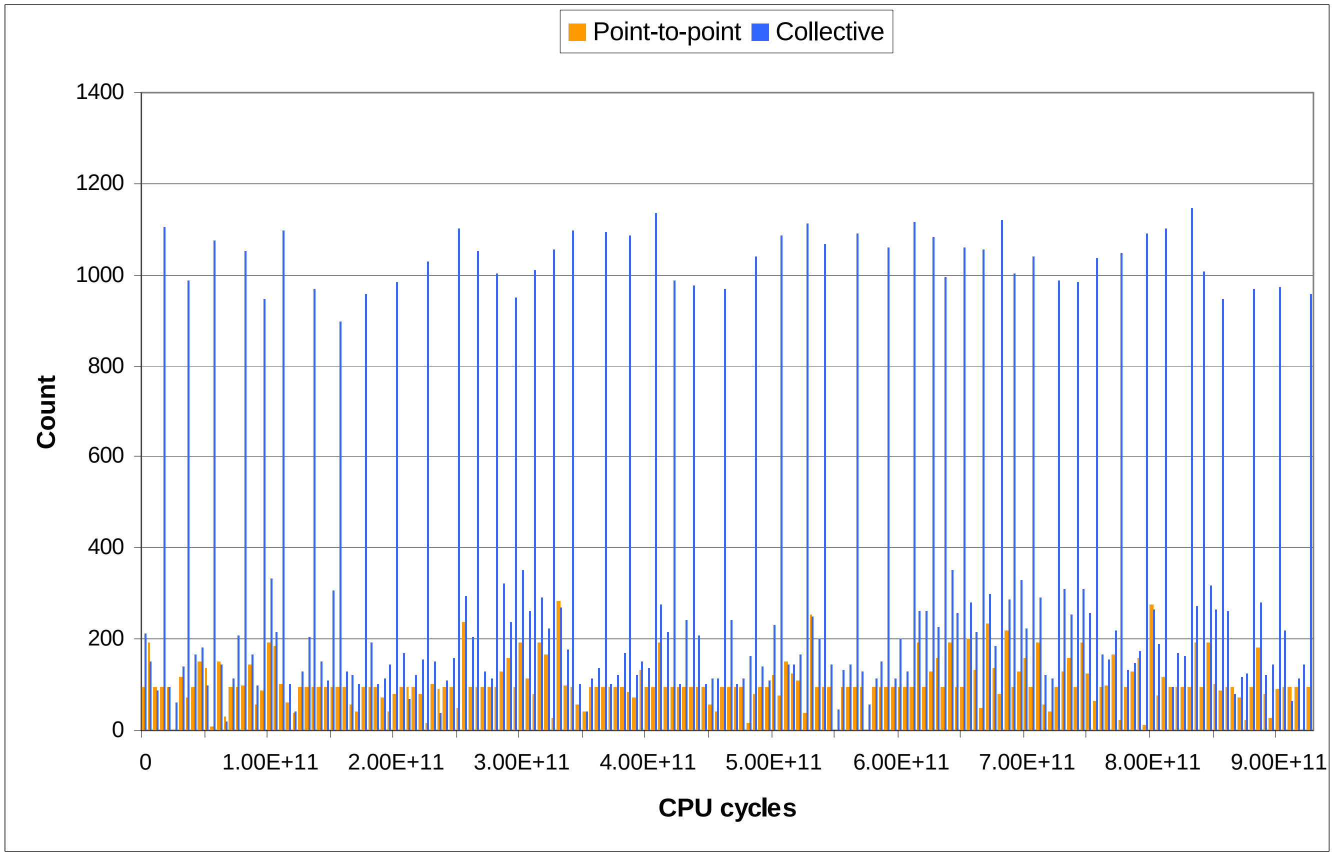
Recursive Coordinate Bisection (RCB)

Domain decomposition kernel written in C by Steve Plimpton (SNL):

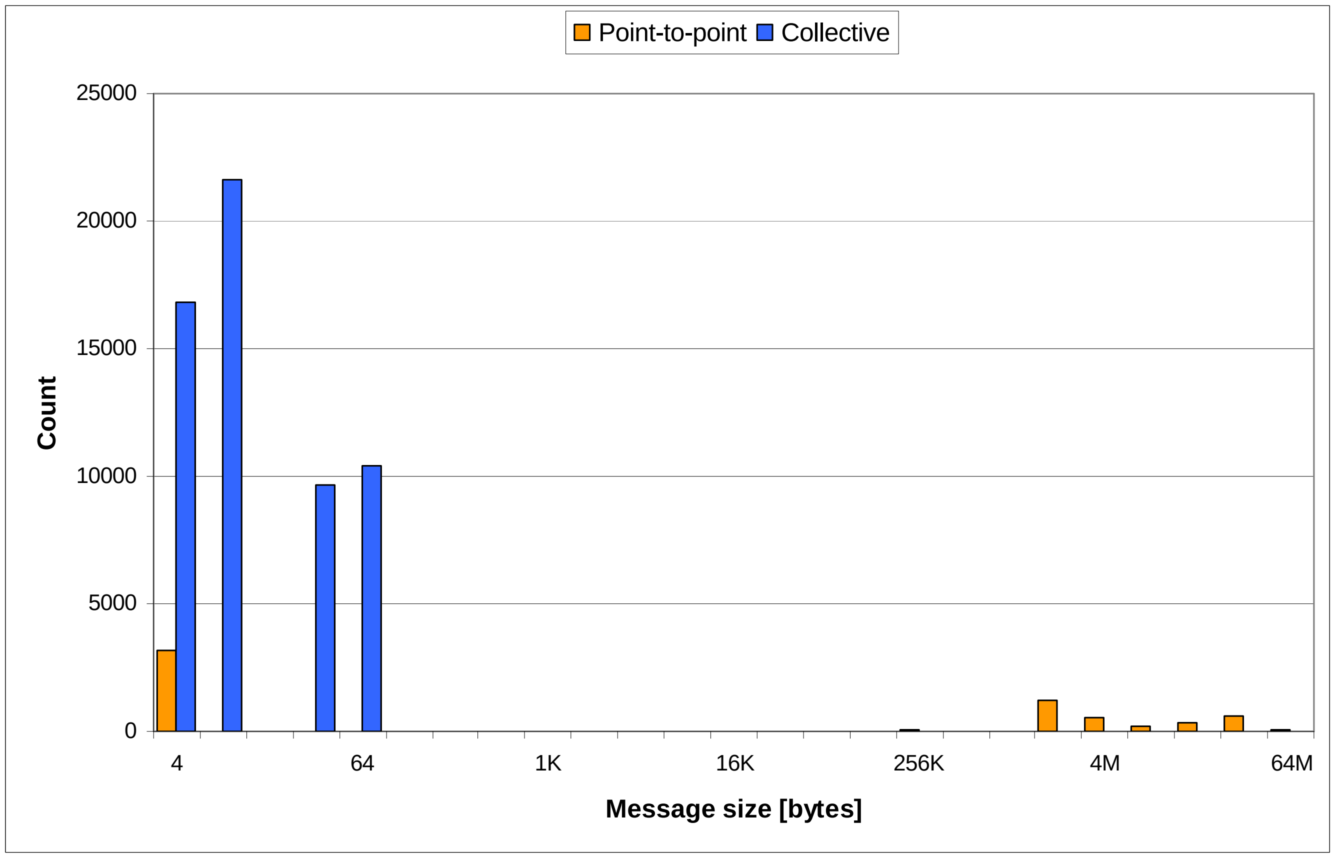
- Recursively subdivides sets of 3-D blocks along their longest edge to evenly distribute the cumulative weight of material points (“dots”) across processors
- Generates random and lattice-aligned test groups of dots
- Weight assignment to dots is optional (distribution by count only)
- Possible to simulate random movements of particles
- Communication bound
  - Reductions and barriers over subdomains with specifically generated communicators
  - Dots propagated through point-to-point calls

RCB: Communication Profile

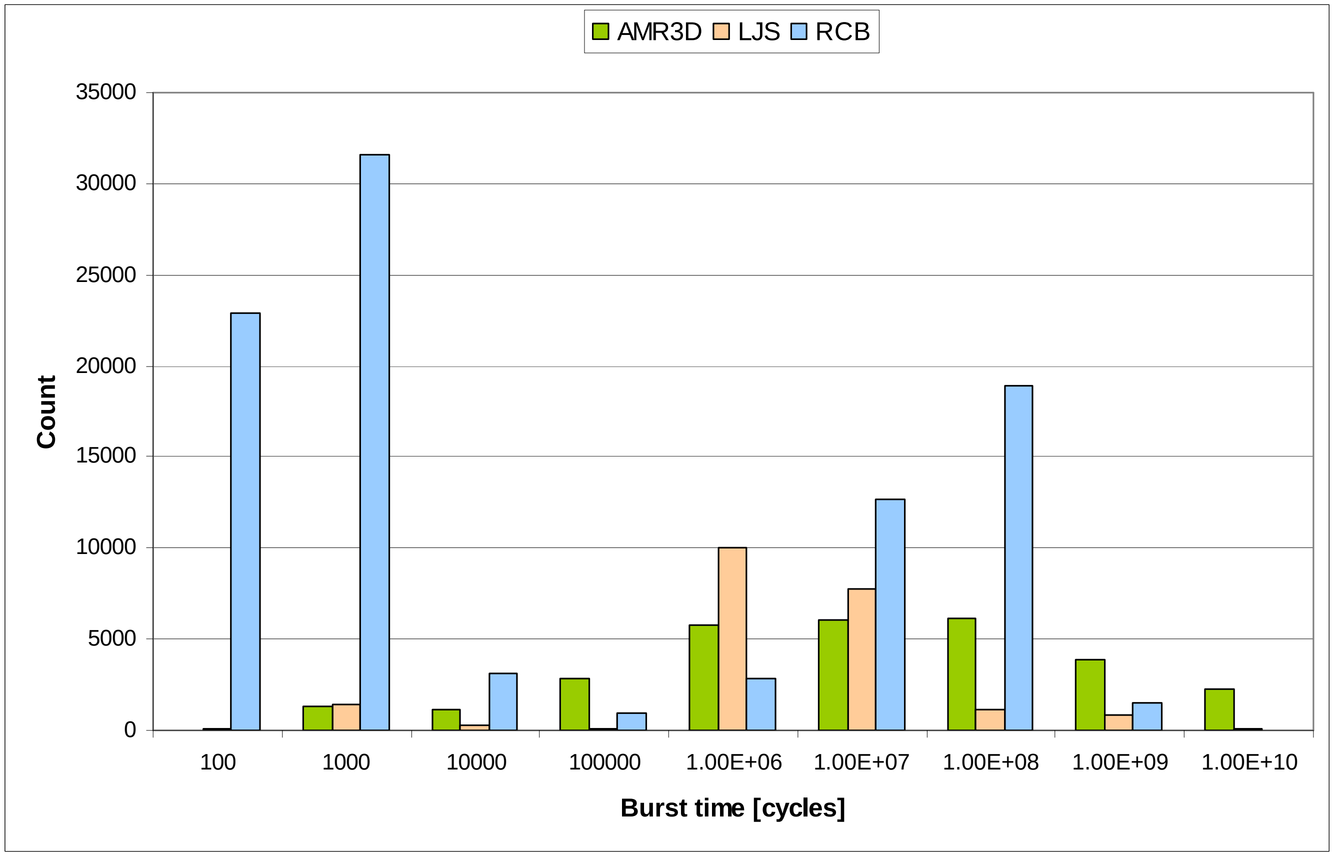
50 time steps on 16 processors, 32 million particles



RCB: Message Size Distribution



Burst time distribution



Near Term Plans

- Create small computational kernels for initialization and statistics gathering in QMC
  - Preparing test runs for IBM to run through the VHDL simulator
- Complete BLUTE tracing for more communications intensive GTC runs and MPI stress tests
- Prepare MHD kernel for BG/L node simulator
- Collect and analyze ETF traces of the remaining applications
- Derive parameters for statistical simulator

Designing a Trace Tool: Desired Characteristics

- Low overhead (both temporal and spatial)
- Freely modifiable and expandable
- Contents of trace files always interpretable by provided tools
- Portable
- Transparent buffering and handling of potentially large amount of data
- Optimized for sequential access, but capable of fast advance through sections of trace file
- Multiprocessor support (trace merging, event matching and ordering)

Expandable Trace Format

Structure:

- Trace is a sequence of files
- Each file includes header, followed by any number of dictionaries and data frames, and a trailer
  - Header describes low-level parameters of trace (endianness, elementary component sizes, version) and, together with trailer, multi-file continuations
  - Dictionaries (token tables) contain definitions of symbols added by the user
  - Frames are sequences of tokens interleaved with actual trace data

ETF: Features

- User-defined token and count field sizes
- Big- and little-endian platform support
- Data types
  - Primitives (integers, FP numbers, strings)
  - Compound (fixed and variable length arrays, records)
- Namespaces
- Hierarchical symbol definitions
- Dictionary generation from human-readable strings
- Event timers
  - TSC register on Intel x86 processors
  - MPI_Wtime
  - gettimeofday()
  - Other?
- Written in C++ (templates for improved performance)

ETF: MPI Support

- All point-to-point communication (MPI-1)
- All collective communication (MPI-1)
- Non-blocking request tracking
- Communicator creation and destruction
- Datatype decoding (requires MPI-2 support)
- Languages: C, Fortran
- Easy instrumentation of applications

MPI Tracing Issues

- Extraction of exact initiation and completion times for non-blocking calls
- Notion of message size in collective communication (e.g., MPI_Barrier vs. MPI_Bcast vs. MPI_Gather)
- Trace interpretation
- Portability of Fortran component

ETF: Memory Reference Tracing

- Tracking of statically and dynamically allocated arrays (identifiers, element sizes, dimensions)
- Tracking of scalar variables
- Read and write accesses to individual scalars and array elements as well as contiguous vectors of elements
- Function calls
- Program execution phases

But:

- Difficult instrumentation (by hand only)

Future Work

- Trace parsing library
- Trace dump tool (equivalent of *bllsute*) with basic filtering capabilities
- Trace merge utility (with global event ordering)
- Further optimization of tracer code
- Better customization of MPI traces
- Statistical analysis tool (min/max, counts, histograms)
- Optional: trace conversion utility (e.g., ETF-BLUTE)
- Optional: trace compaction tool